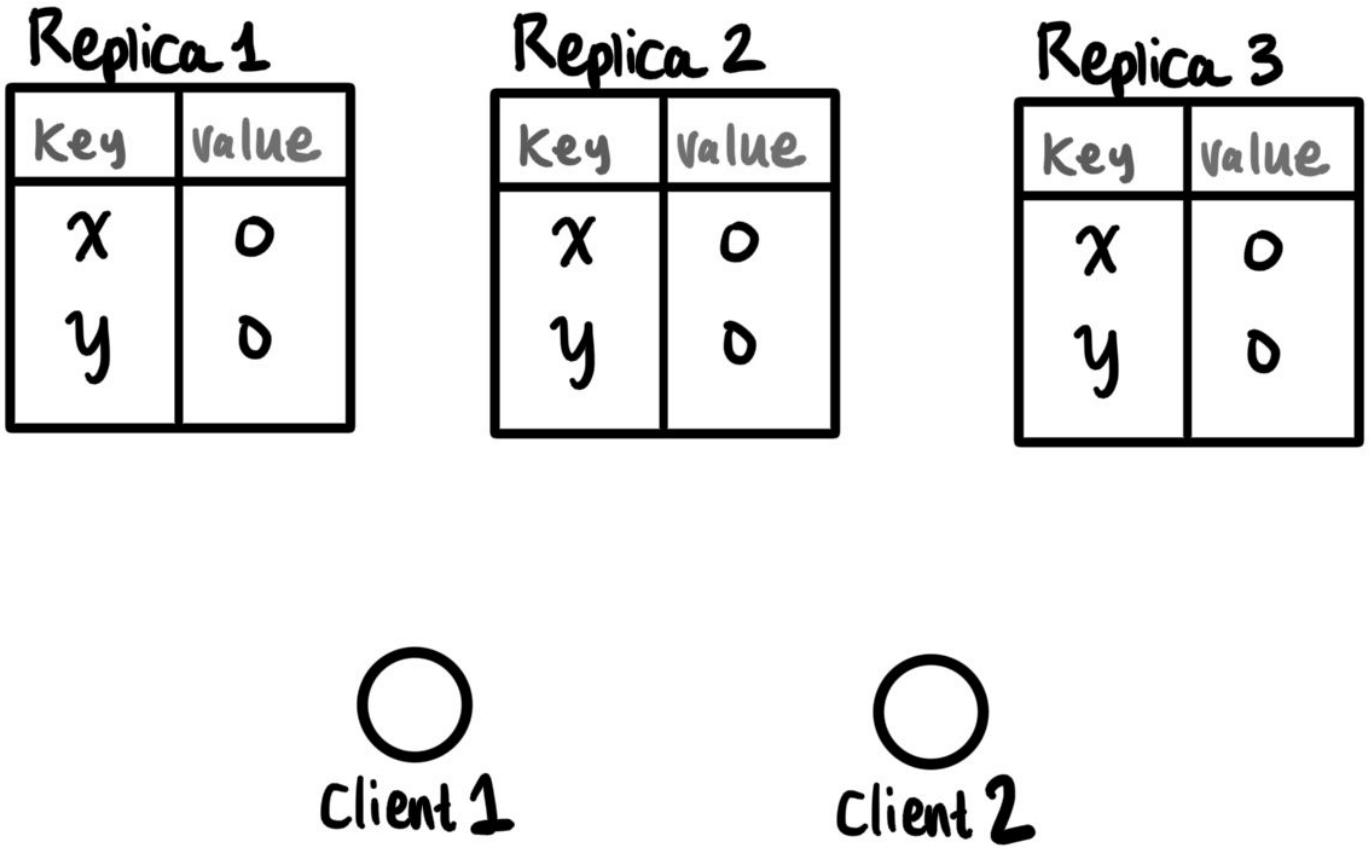
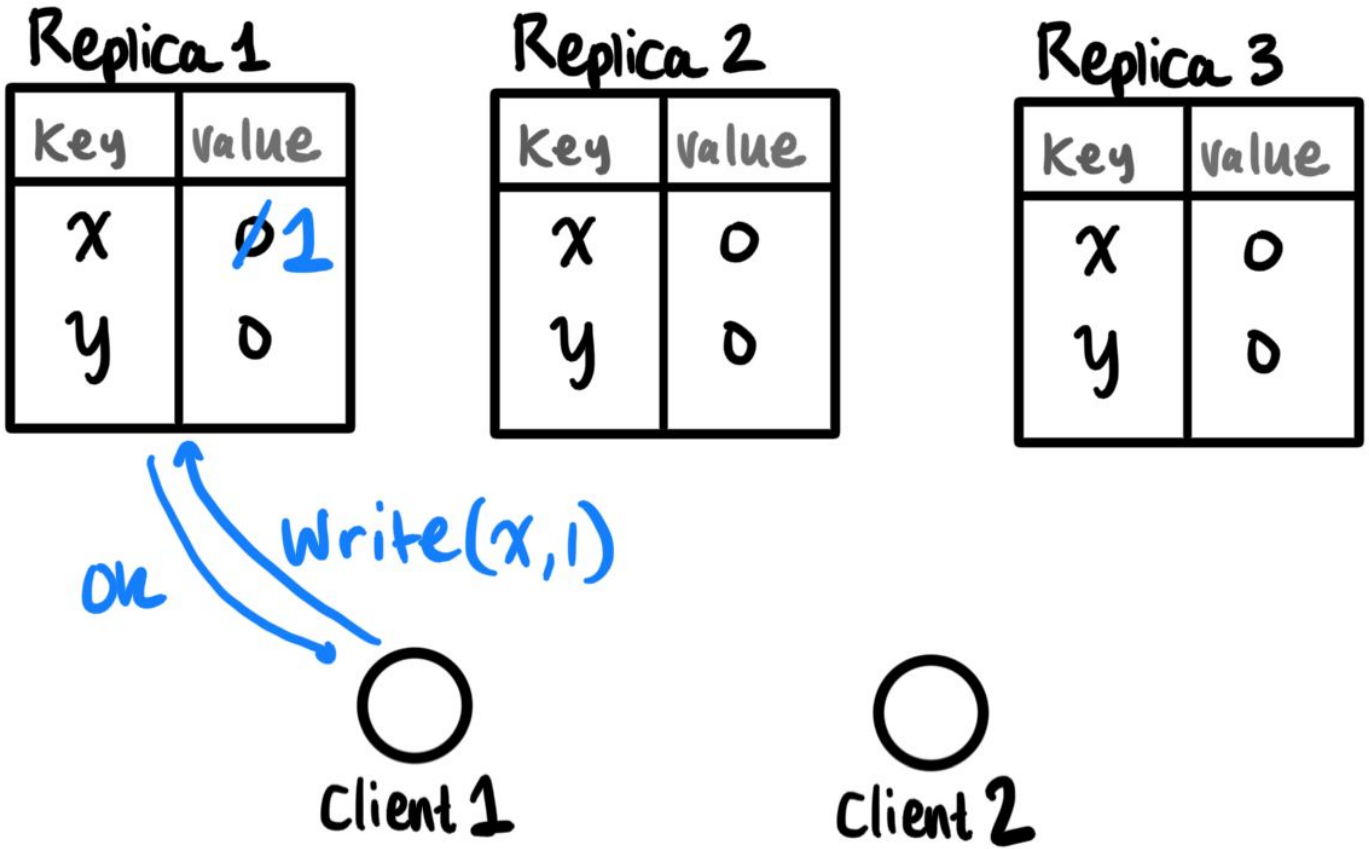
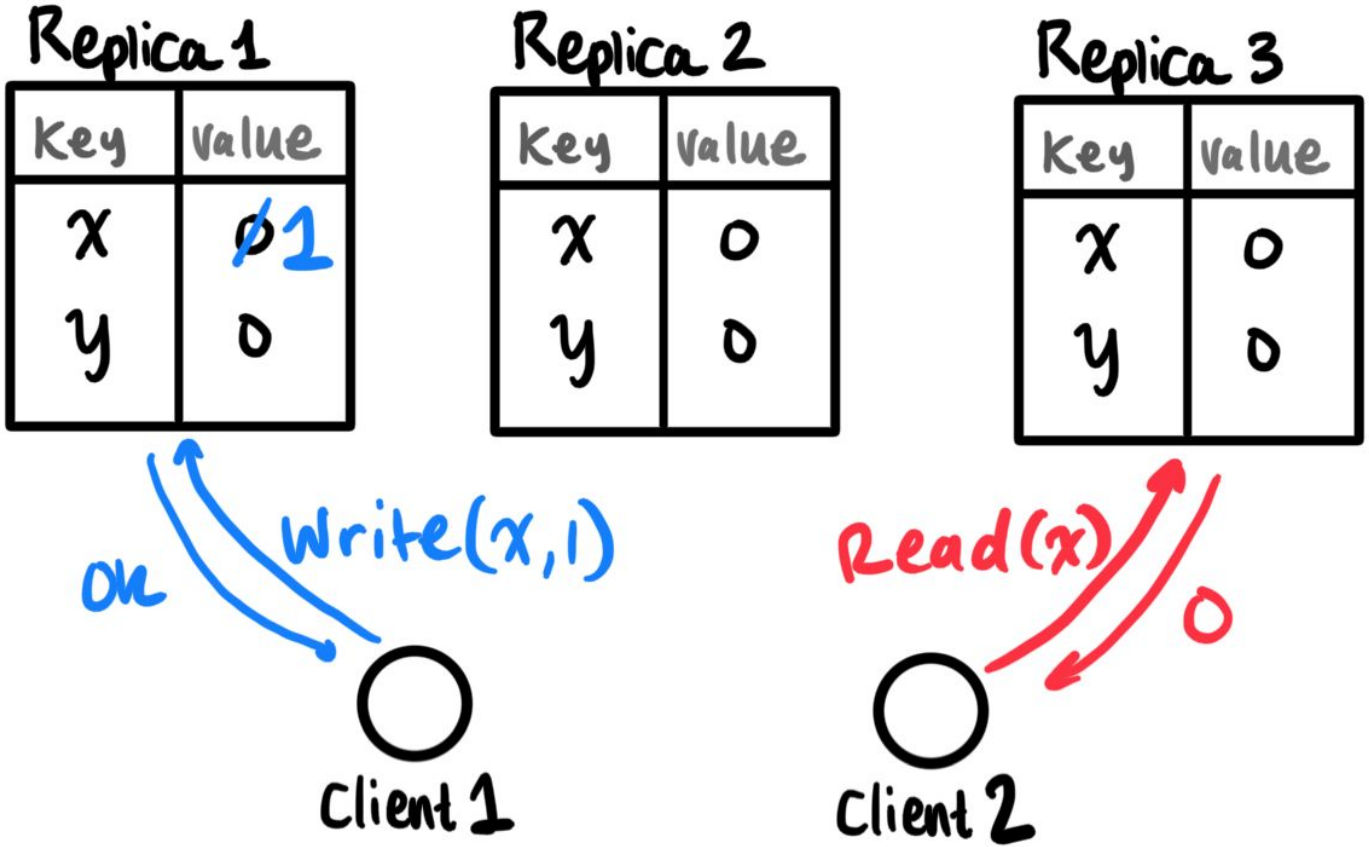
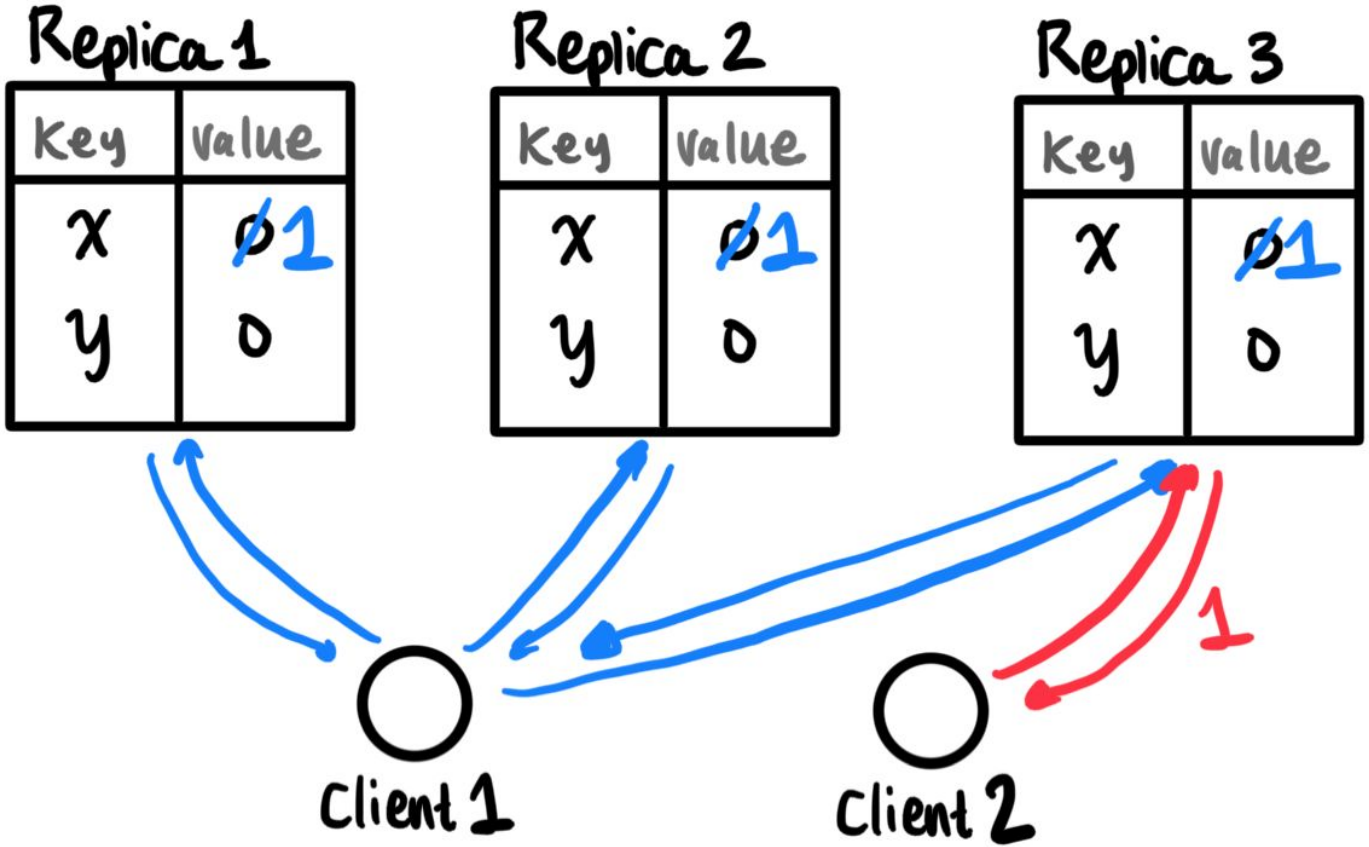
Michael Whittaker, Aleksey Charapko, Joseph M. Hellerstein, Heidi Howard, Ion Stoica


Replica 1
Key
value
x
0
y
0
Replica 2
Key
value
x
0
y
0
Replica 3
Key
value
x
0
y
0
Client 1
Client 2


Replica 1
Key
Value
x
0 1
y
0
Replica 2
Key
Value
x
0
y
0
Replica 3
Key
Value
x
0
y
0
Write(x,1)
ok
Client 1
Client 2


Replica 1
Key
Value
x
0
1
y
0
Replica 2
Key
Value
x
0
y
0
Replica 3
Key
Value
x
0
y
0
Ok
Write(x,1)
Read(x)
0
Client 1
Client 2


Replica 1
Key
Value
x
0
1
y
0
Replica 2
Key
Value
x
0
1
y
0
Replica 3
Key
Value
x
0
1
y
0
1
Client 1
Client 2

Given a set $X = \{x_1, \ldots, x_n\}$, a **read-write quorum system** over X is a pair $Q = (R, W)$ where

Given a set $X = \{x_1, \ldots, x_n\}$, a **read-write quorum system** over X is a pair $Q = (R, W)$ where

1. R is a set of subsets of X called read quorums.

Given a set $X = \{x_1, \ldots, x_n\}$, a **read-write quorum system** over X is a pair $Q = (R, W)$ where

1. R is a set of subsets of X called read quorums.
2. W is a set of subsets of X called write quorums.

Given a set $X = \{x_1, \ldots, x_n\}$, a **read-write quorum system** over X is a pair $Q = (R, W)$ where

1. R is a set of subsets of X called read quorums.
2. W is a set of subsets of X called write quorums.
3. Every read quorum intersects every write quorum: $\forall r \in R. \forall w \in W. r \cap w \neq \emptyset$

The majority quorum system.

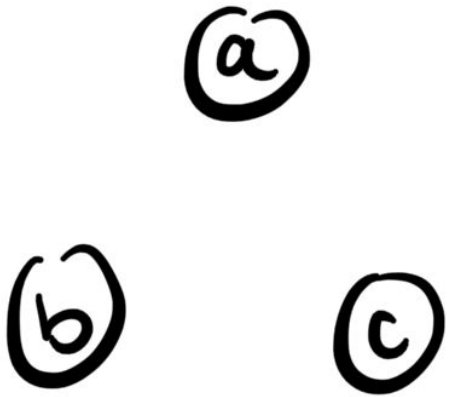
$X = \{a, b, c\}$

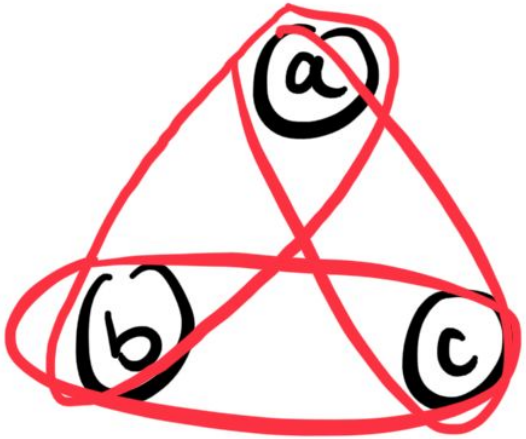
Read quorums

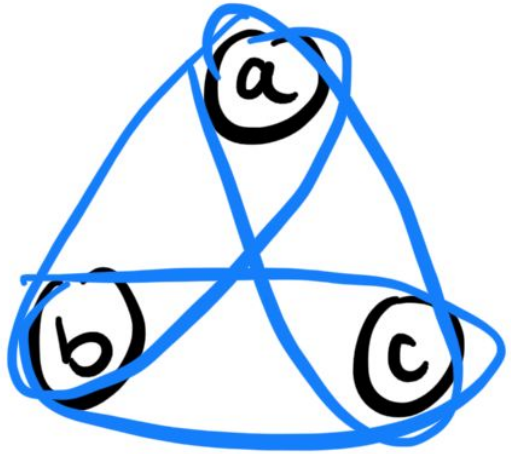
$R = \{ab, bc, ac\}$

Write quorums

$W = \{ab, bc, ac\}$







The grid quorum system.

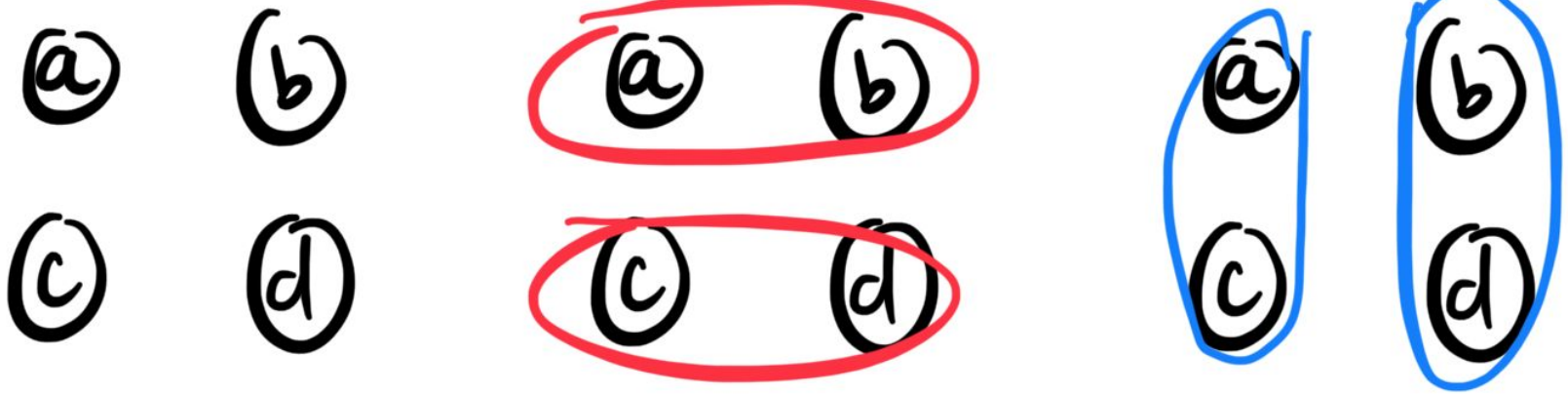
$X = \{a, b, c, d\}$

Read quorums

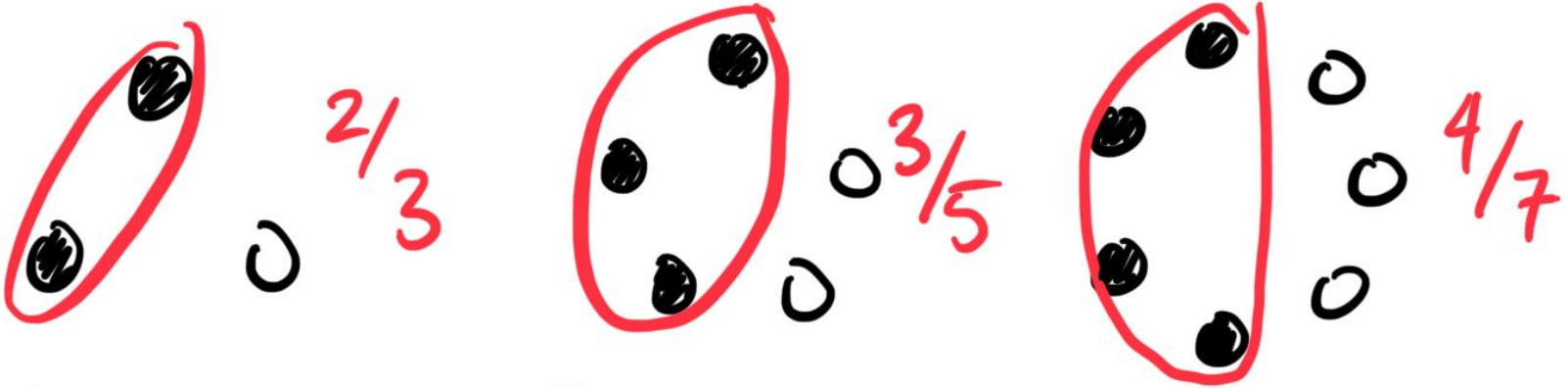
$R = \{ab, cd\}$

Write quorums

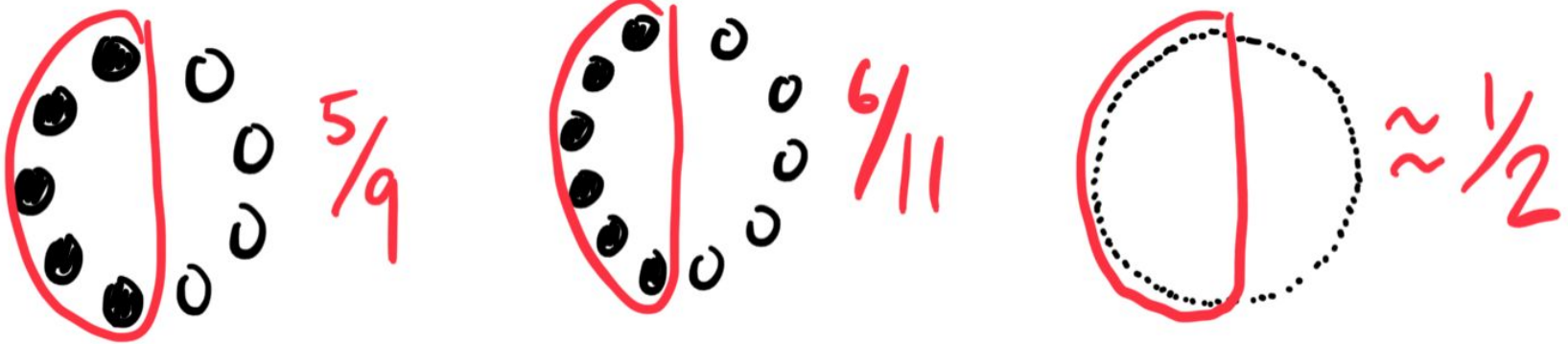
$W = \{ac, bd\}$

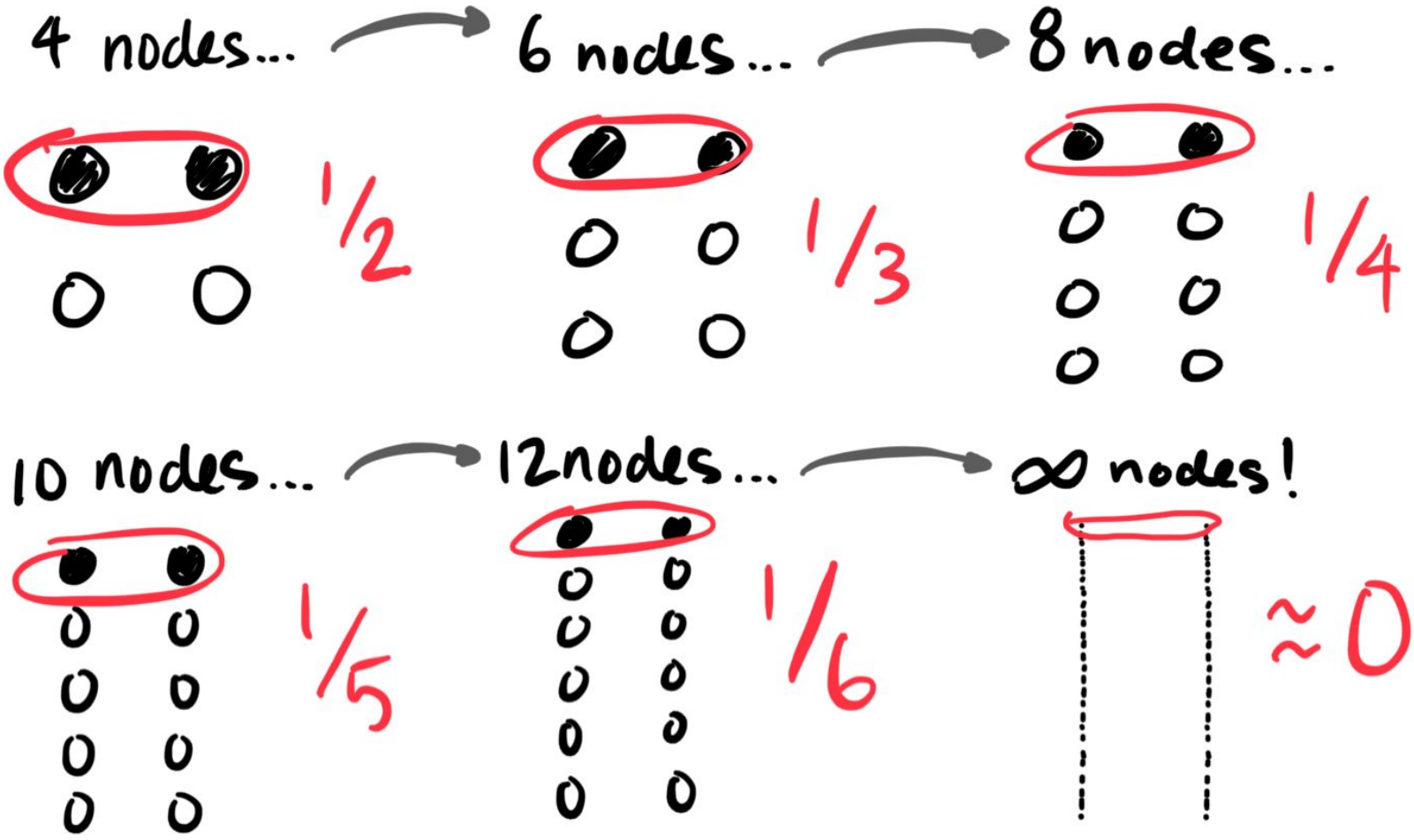


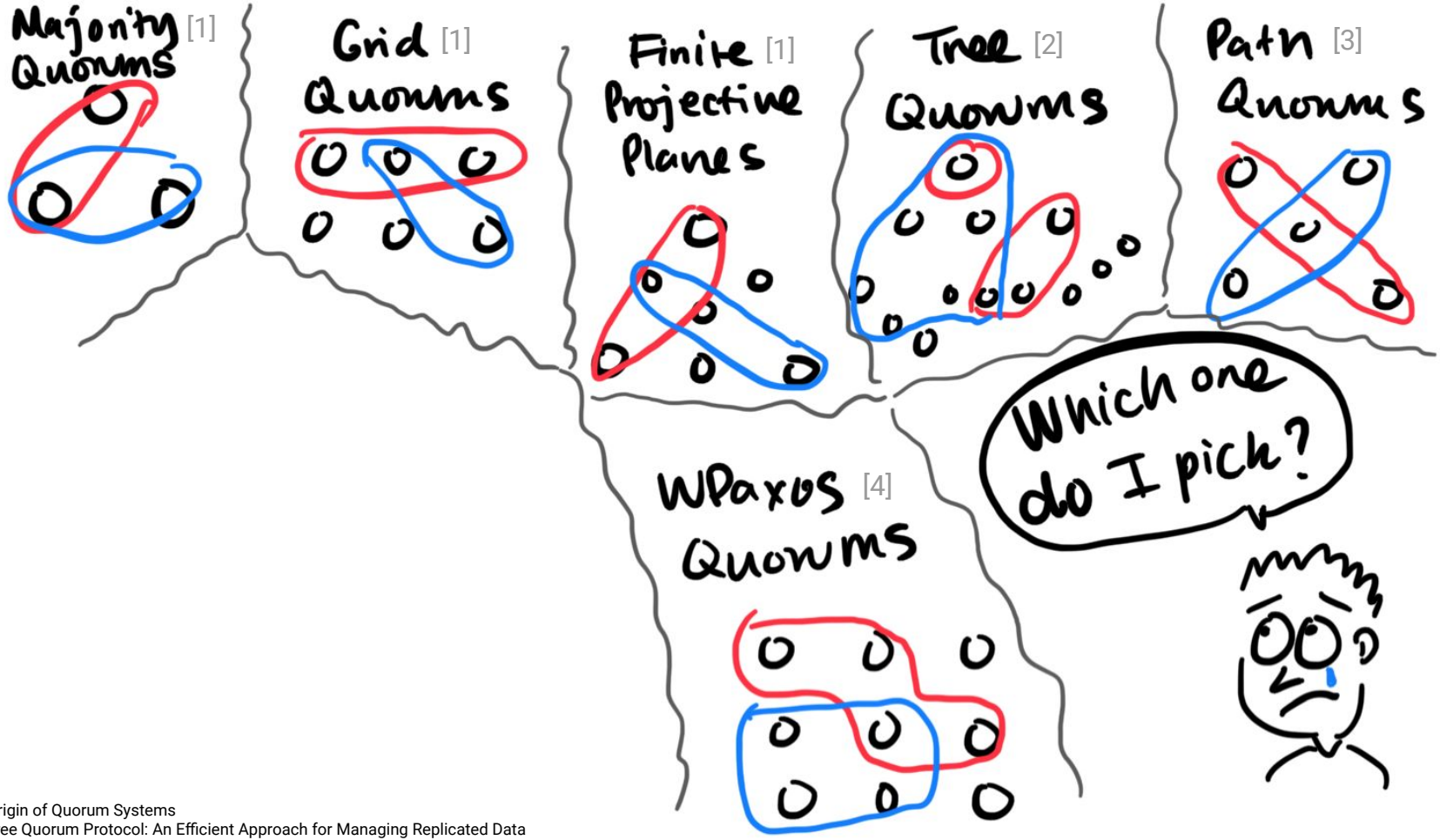
3 nodes... → 5 nodes... → 7 nodes...



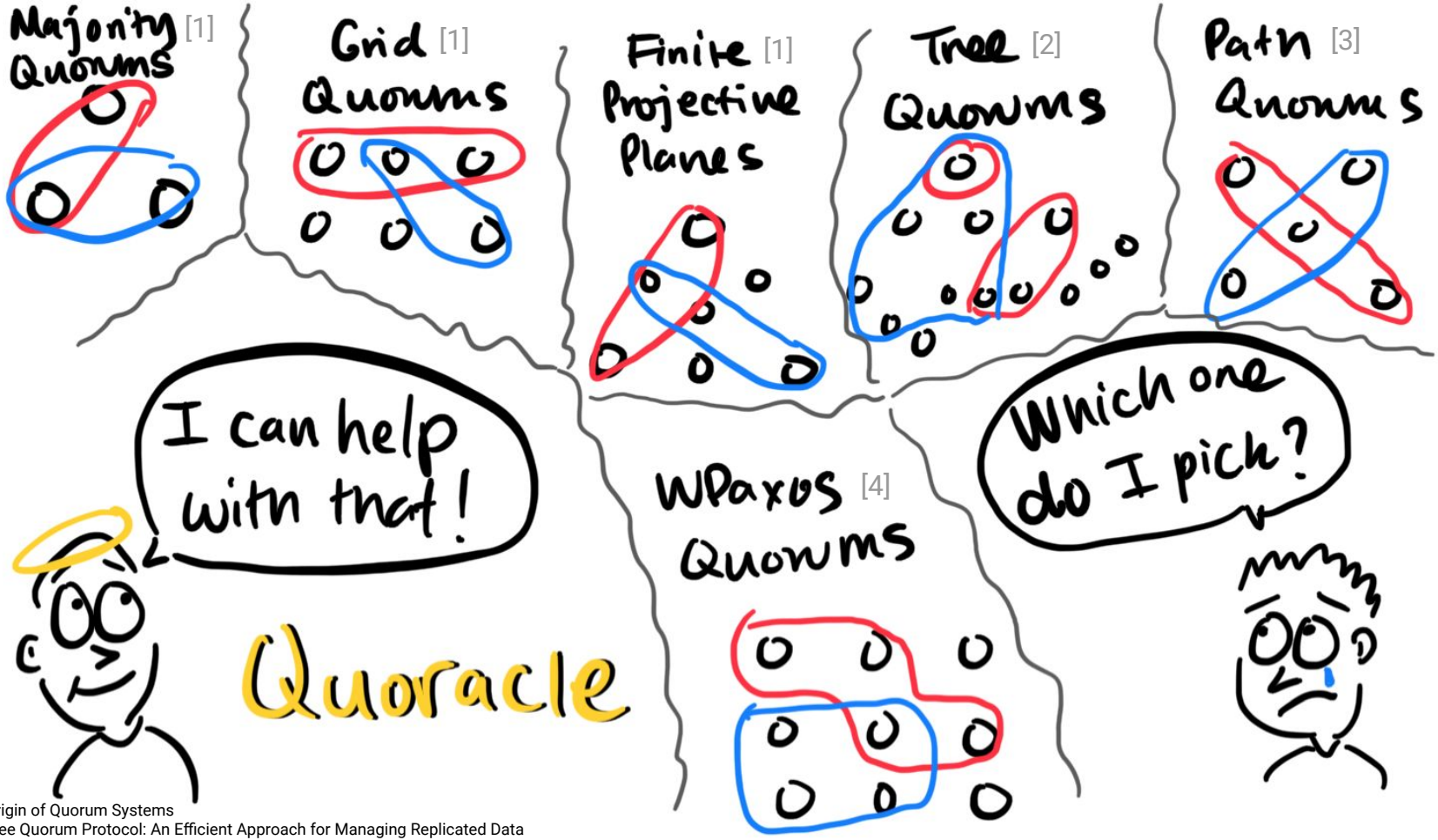
9 nodes... → 11 nodes... → ∞ nodes!




4 nodes...
1/2
6 nodes...
1/3
8 nodes...
1/4
10 nodes...
1/5
12nodes...
1/6
∞ nodes!
≈0



[1]: The Origin of Quorum Systems
[2]: The Tree Quorum Protocol: An Efficient Approach for Managing Replicated Data
[3]: The Load, Capacity, and Availability of Quorum Systems
[4]: WPaxos: Wide Area Network Flexible Consensus



[1]: The Origin of Quorum Systems
[2]: The Tree Quorum Protocol: An Efficient Approach for Managing Replicated Data
[3]: The Load, Capacity, and Availability of Quorum Systems
[4]: WPaxos: Wide Area Network Flexible Consensus

New quorum system theory

+

Python library/tool

$$f_r\left(\sum_{\{r \in R \mid x \in R\}} p_r\right) +$$

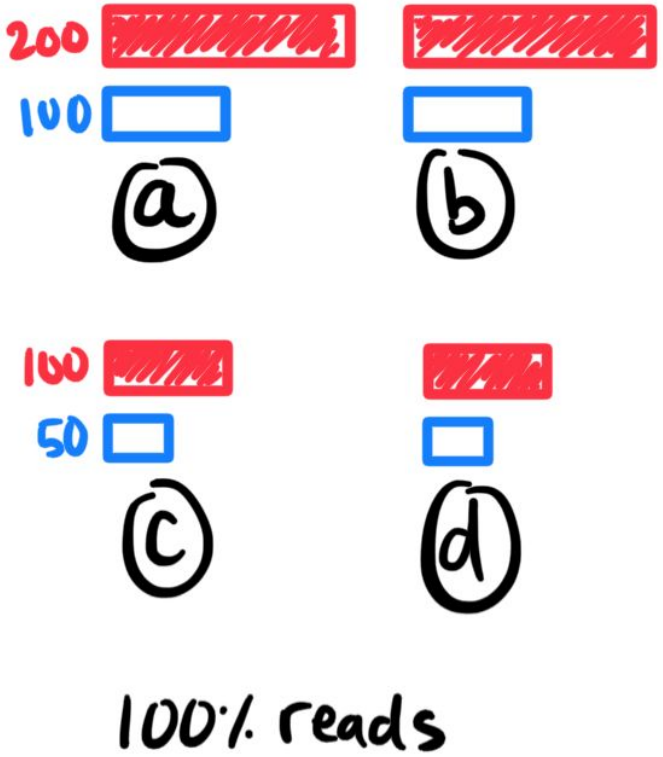
$$(1-f_r)\left(\sum_{\{w \in W \mid x \in w\}} p_w\right)$$

```
from quoracle import *

a = Node('a')
v = Node('v')
t = Node('t')
n = Node('n')

qs = QuorumSystem(a*v*a + t*a*n)
```

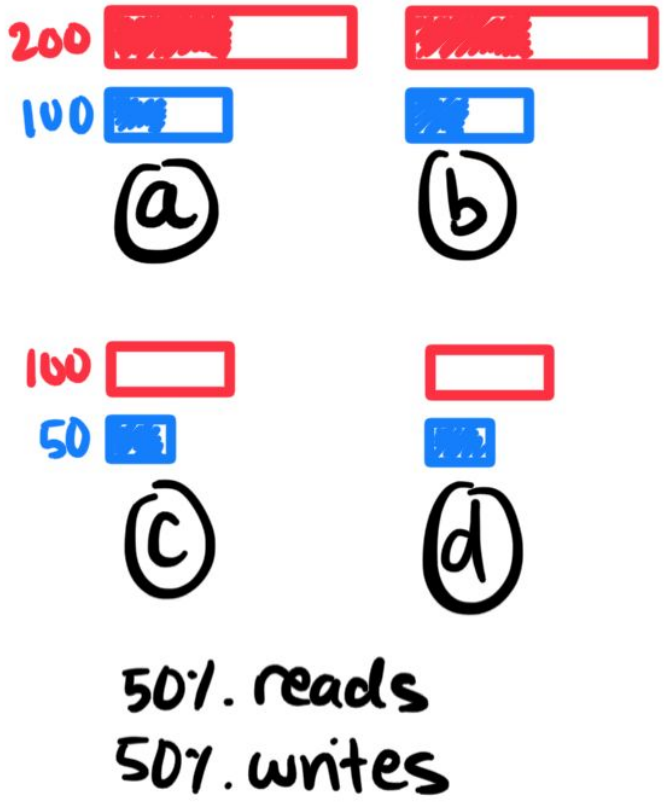
```python
from quoracle import *

a = Node('a')
b = Node('b')
c = Node('c')
majority = QuorumSystem(reads=a*b + b*c + a*c)

print(majority.fault_tolerance()) # 1
print(majority.load(read_fraction=1)) # 2/3
print(majority.capacity(read_fraction=1)) # 3/2
```

```
from quoracle import *

a = Node('a', write_cap=100, read_cap=200)
b = Node('b', write_cap=100, read_cap=200)
c = Node('c', write_cap=50, read_cap=100)
d = Node('d', write_cap=50, read_cap=100)
grid = QuorumSystem(reads=a*b + c*d)

print(grid.capacity(read_fraction=1)) # 300
print(grid.capacity(read_fraction=0.5)) # 200
print(grid.capacity(read_fraction=0)) # 100
```

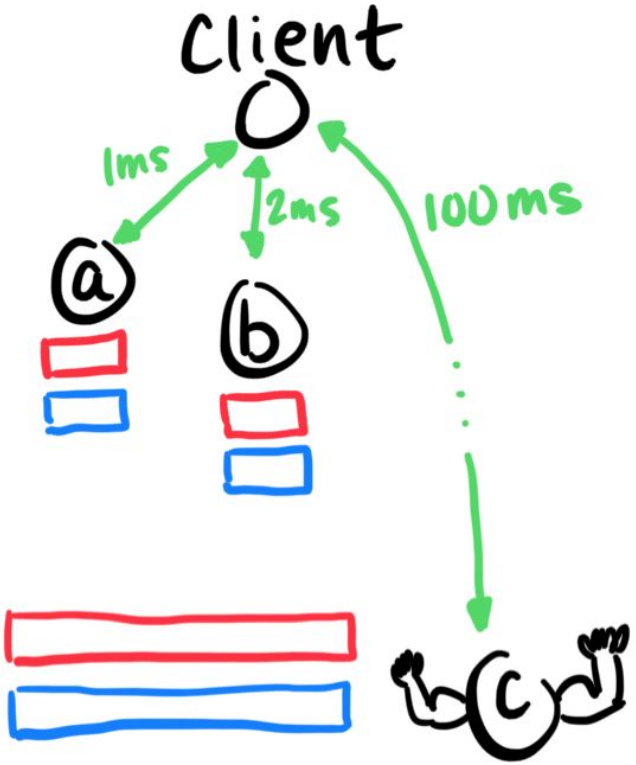
```python
from quoracle import *

a = Node('a', write_cap=100, read_cap=200)
b = Node('b', write_cap=100, read_cap=200)
c = Node('c', write_cap=50, read_cap=100)
d = Node('d', write_cap=50, read_cap=100)
grid = QuorumSystem(reads=a*b + c*d)

print(grid.capacity(read_fraction=1)) # 300
print(grid.capacity(read_fraction=0.5)) # 200
print(grid.capacity(read_fraction=0)) # 100
```



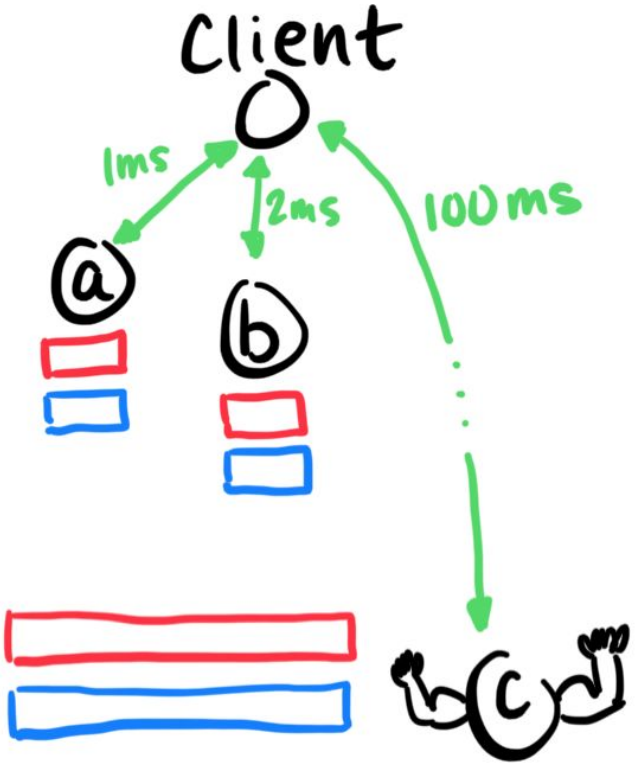
```python
from quoracle import *

a = Node('a', write_cap=100, read_cap=200)
b = Node('b', write_cap=100, read_cap=200)
c = Node('c', write_cap=50, read_cap=100)
d = Node('d', write_cap=50, read_cap=100)
grid = QuorumSystem(reads=a*b + c*d)

print(grid.capacity(read_fraction=1)) # 300
print(grid.capacity(read_fraction=0.5)) # 200
print(grid.capacity(read_fraction=0)) # 100
```



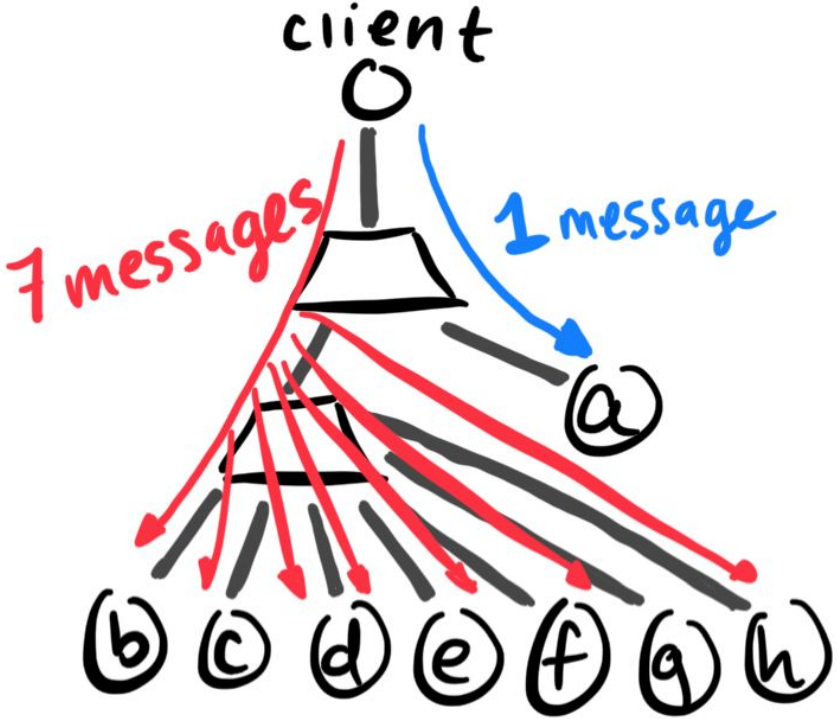
Latency



Latency

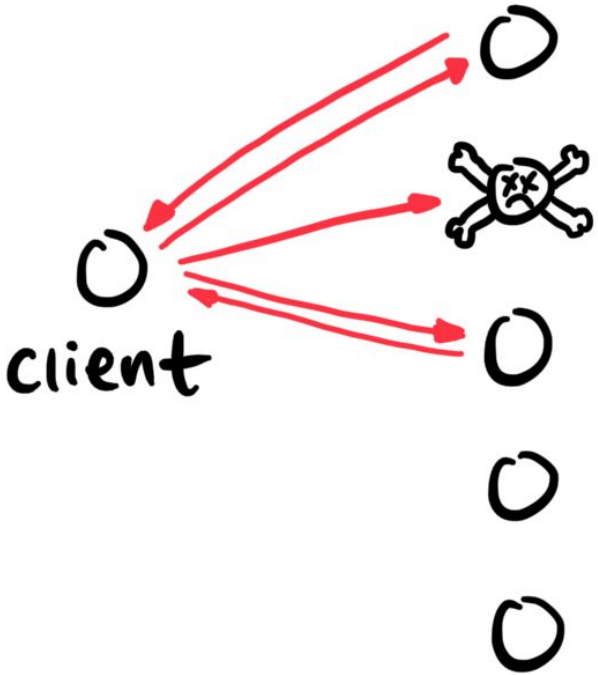


Network Load

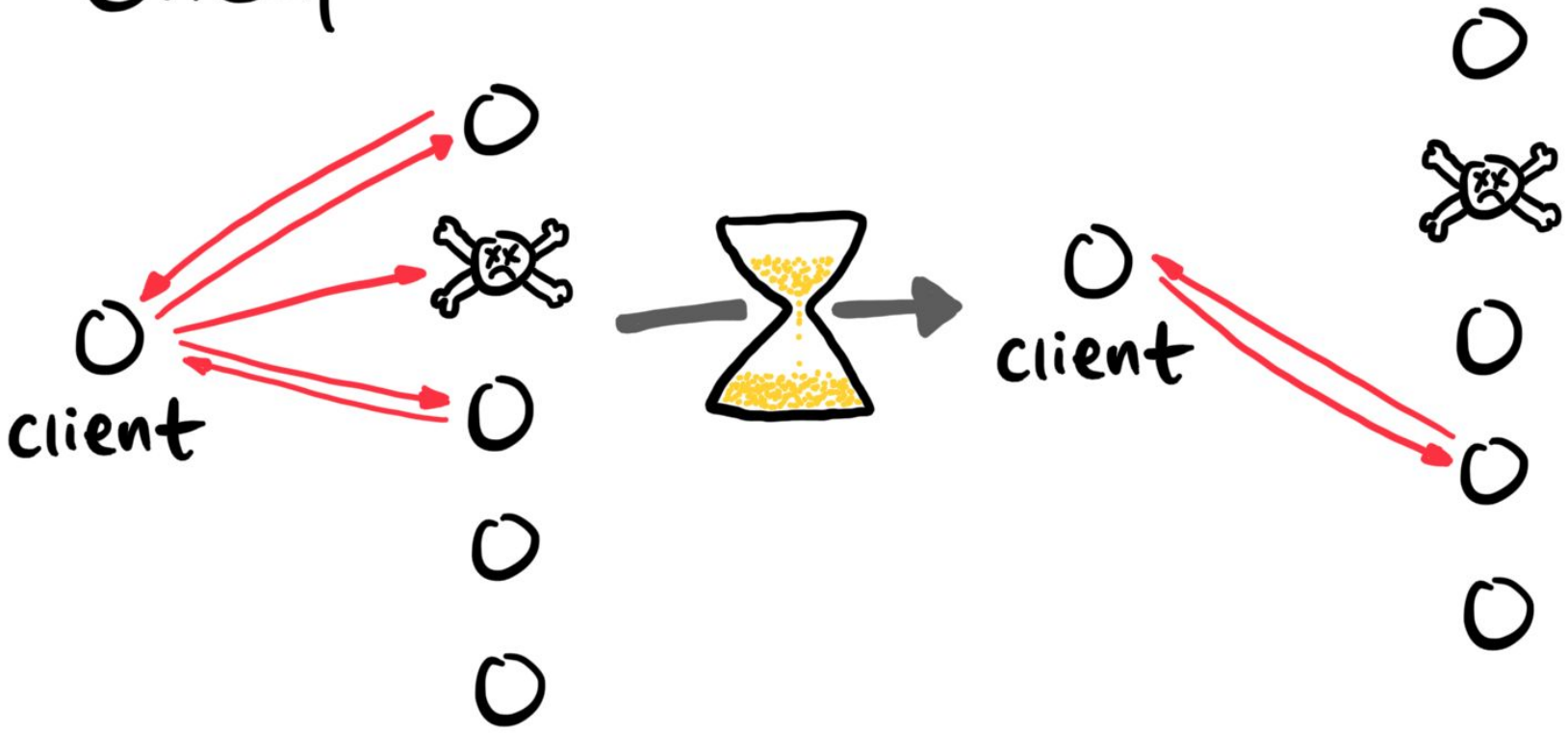


```python
import datetime

def seconds(x: int) -> datetime.timedelta:
    return datetime.timedelta(seconds=x)

a = Node('a', latency=seconds(1))
b = Node('b', latency=seconds(2))
c = Node('c', latency=seconds(3))
d = Node('d', latency=seconds(4))
e = Node('e', latency=seconds(5))
f = Node('f', latency=seconds(6))
grid = QuorumSystem(reads=a*b*c + d*e*f)

grid.latency(read_fraction=0.5, optimize='latency') # 0:00:03.500000
grid.network_load(read_fraction=0.5, optimize='network') # 2.5
```

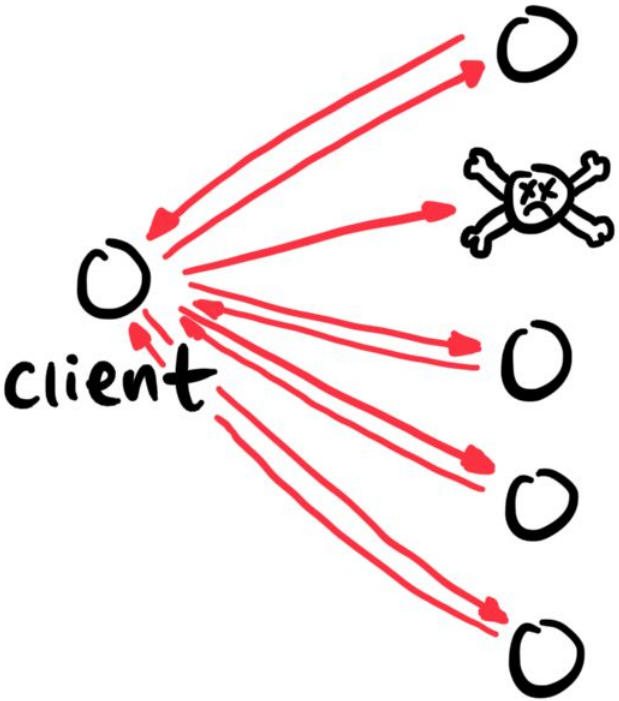
Cheap

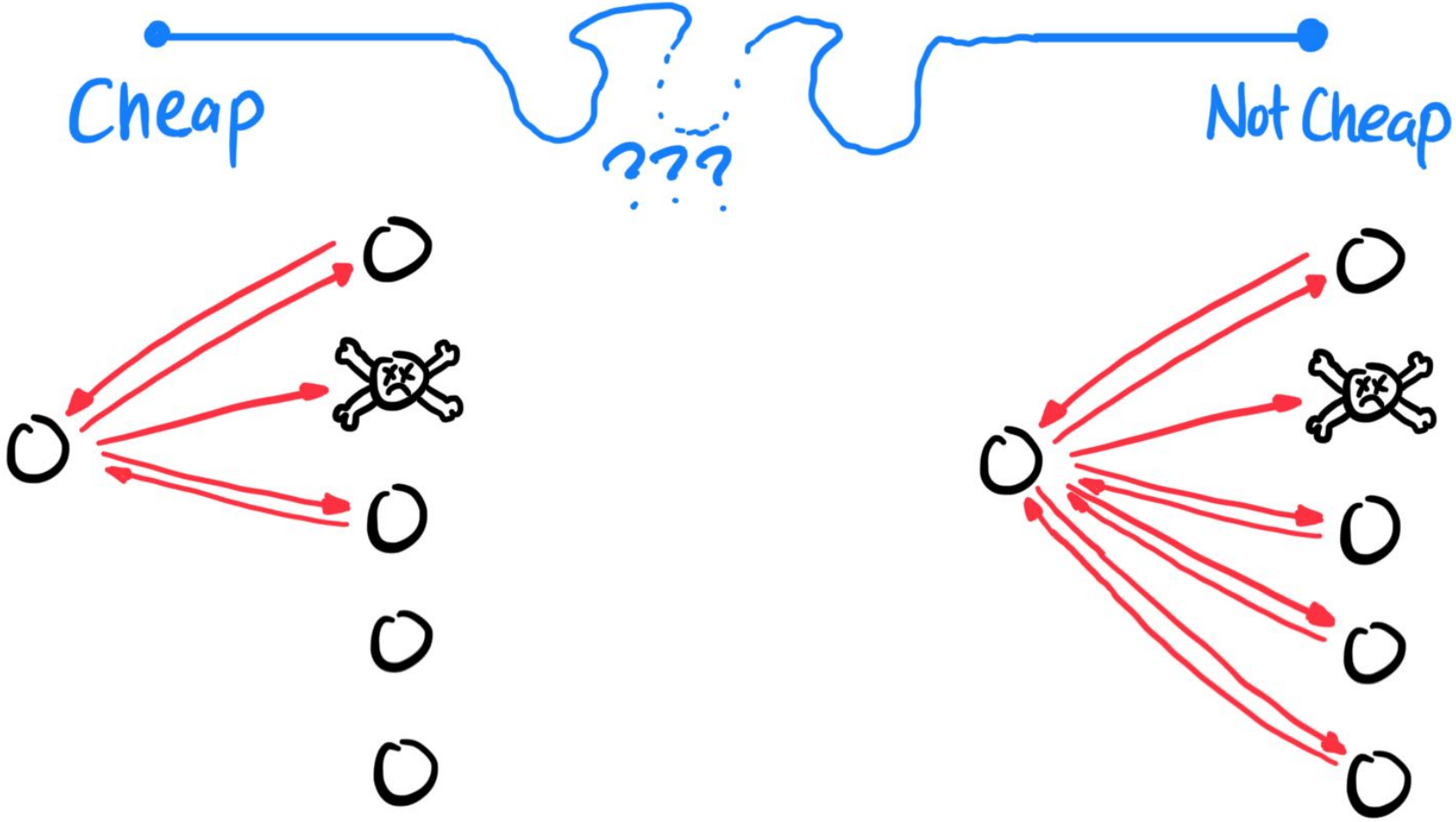


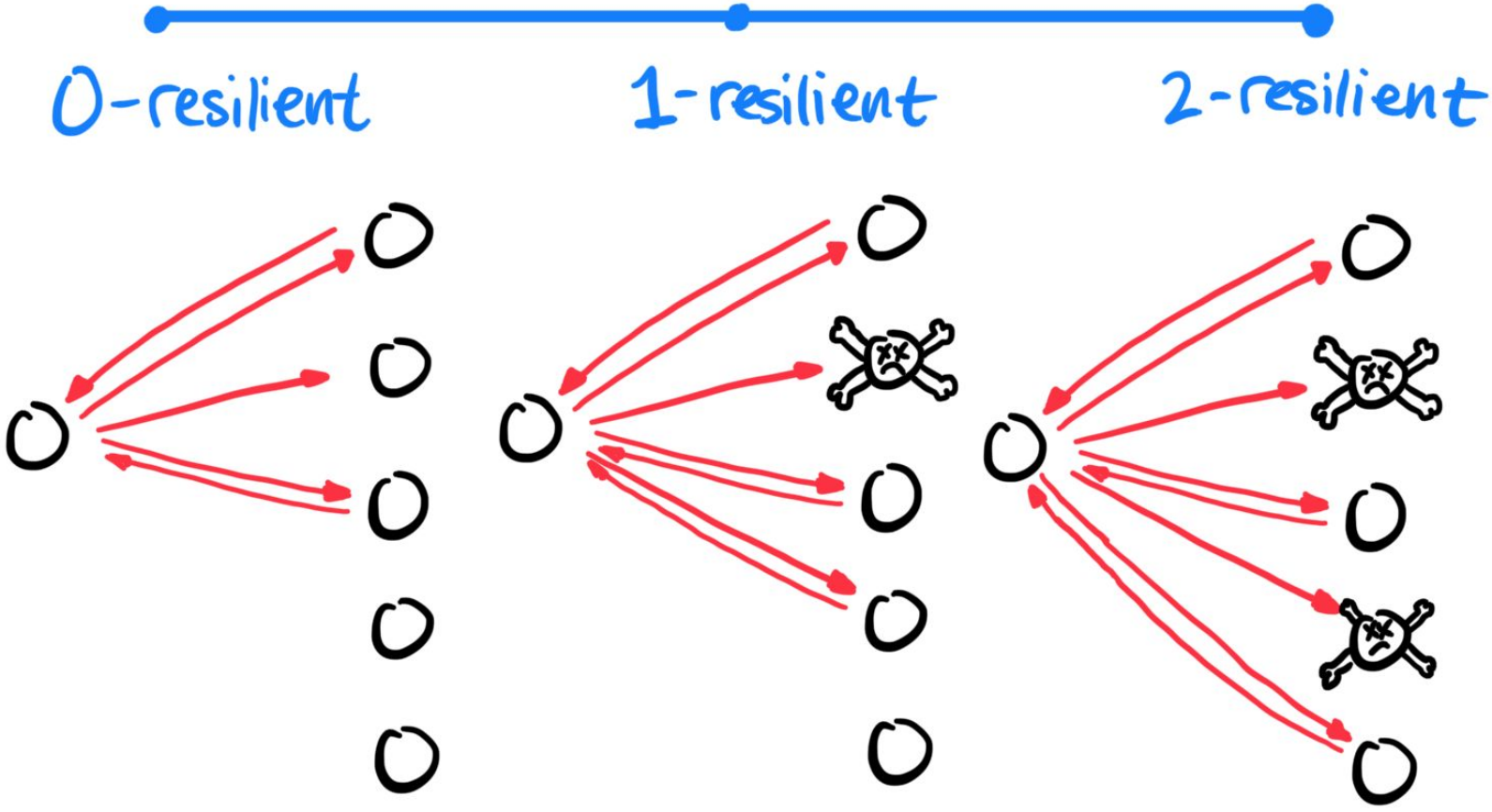
Cheap



Not Cheap

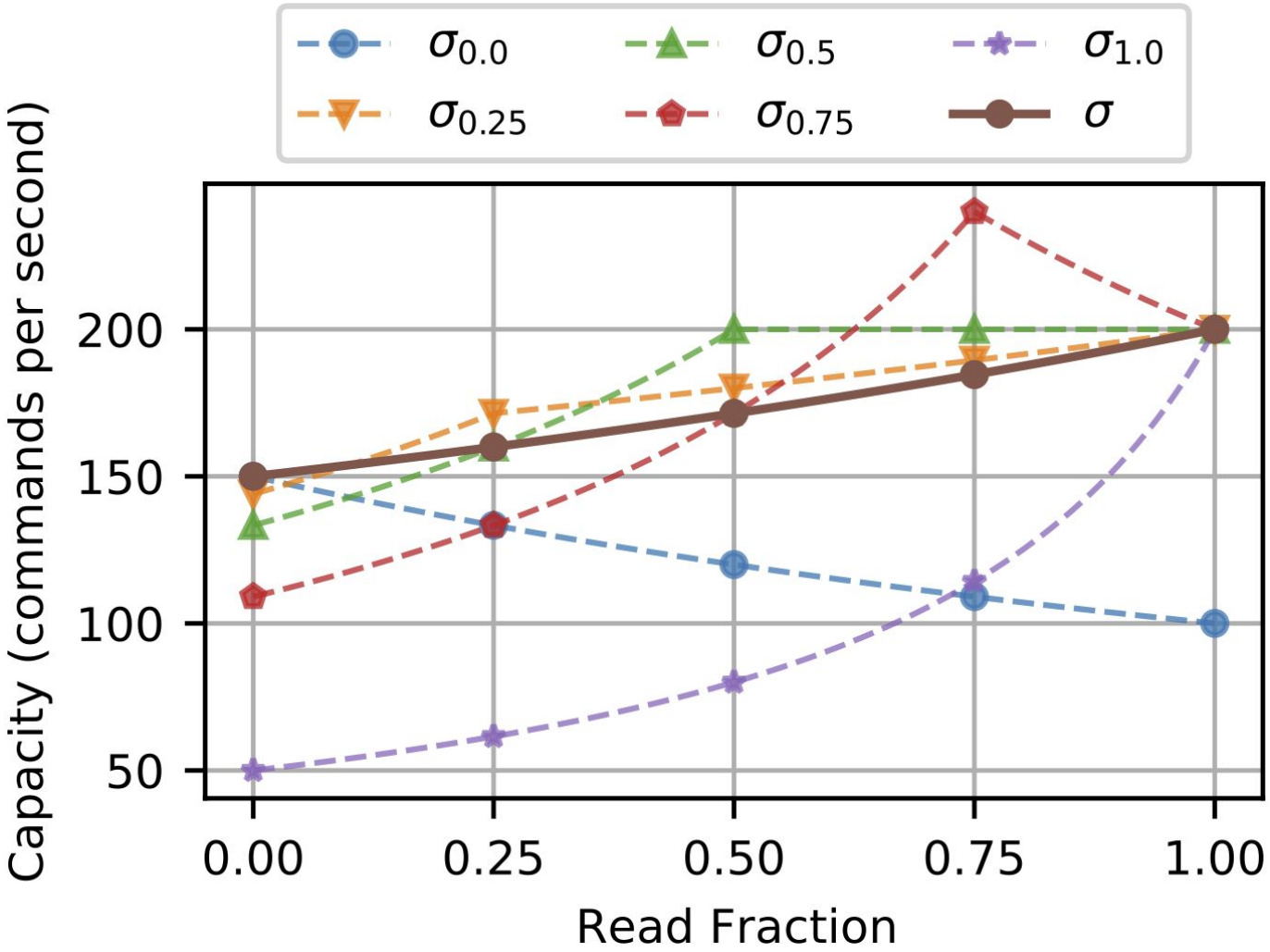


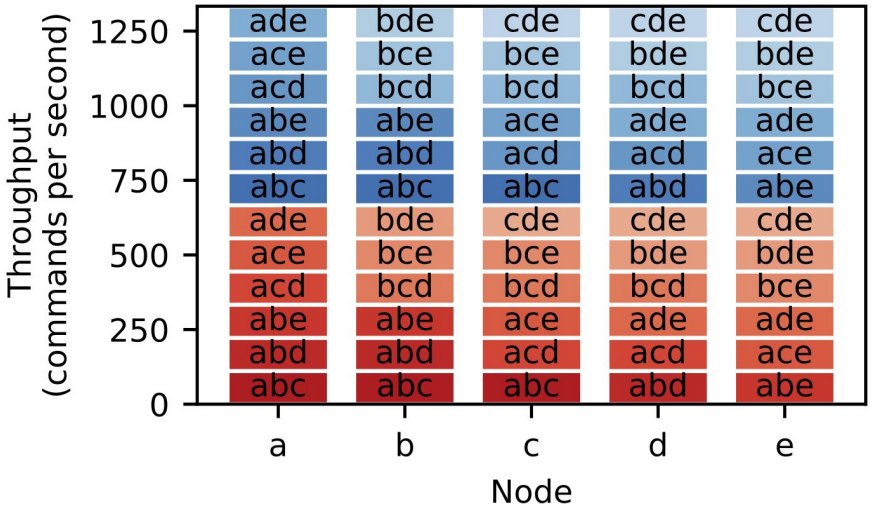

Cheap
???
Not Cheap

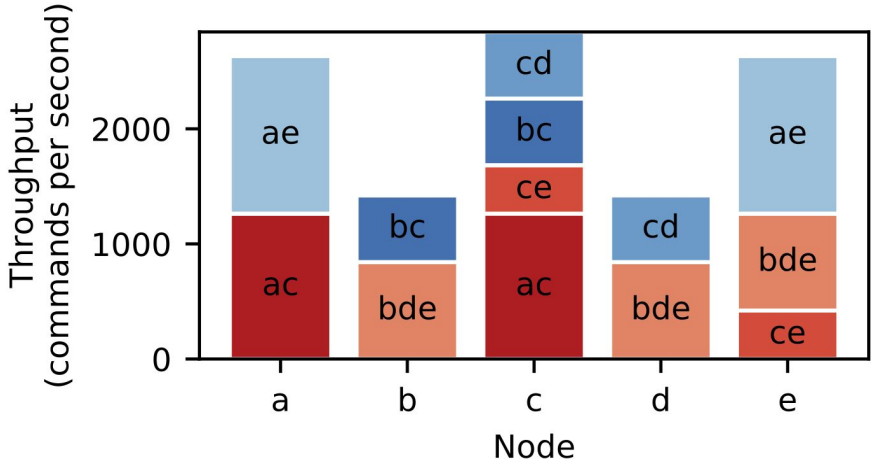

0-resilient
1-resilient
2-resilient

```
quorum_system.capacity(read_fraction=0, f=1)
```

```
quorum_system, strategy = search(
    nodes=[a, b, c, d, e, f],
    resilience=1,
    f=1,
    read_fraction=0.75,
    optimize='load',
    latency_limit=seconds(4),
    network_limit=4,
    timeout=seconds(60),
)
```


$\sigma_{0.0}$
$\sigma_{0.5}$
$\sigma_{1.0}$
$\sigma_{0.25}$
$\sigma_{0.75}$
σ
Capacity (commands per second)
200
150
100
50
0.00
0.25
0.50
0.75
1.00
Read Fraction


Throughput
(commands per second)
1250
1000
750
500
250
0
ade bde cde cde cde
ace bce bce bde bde
acd bcd bcd bcd bce
abe abe ace ade ade
abd abd acd acd ace
abc abc abc abd abe
ade bde cde cde cde
ace bce bce bde bde
acd bcd bcd bcd bce
abe abe ace ade ade
abd abd acd acd ace
abc abc abc abd abe
a
b
c
d
e
Node


Throughput
(commands per second)
2000
1000
0
ae
ac
bc
bde
cd
ce
ac
bde
cd
bde
ae
bde
ce
a
b
c
d
e
Node

github.com/mwhittaker/quoracle

```
pip install quoracle
```